On Intersection Types and Probabilistic Lambda-Calculi*

Flavien Breuvart† Ugo Dal Lago‡

November 19, 2018

Abstract

We define two intersection type systems for the pure, untyped, probabilistic λ-calculus, and prove that type derivations precisely reflect the probability of convergence of the underlying term. We first define a simple system of *oracle* intersection types in which derivations are annotated by binary strings and the probability of termination can be computed by combining *all* the different possible annotations. Although inevitable due to recursion theoretic limitations, the fact that (potentially) infinitely many derivations need to be considered is of course an issue when seeing types as a verification methodology. We thus develop a more complex system: the *monadic* intersection type system. In this second system, the probability of termination of a term is shown to be the *least upper bound* of the weights of its type derivations.

1 Introduction

Interactions between computer science and probability theory are pervasive and fruitful. Probability theory offers models that enable system abstraction, but it also suggests a new model *of computation*, which is central to randomised computation [32] and cryptography [17]. All this has stimulated the study of probabili-stic computational models and programming languages: probabilistic variations on automata [12, 33], Turing machines [35, 15], and the λ-calculus [34], have indeed been introduced and studied since the early days of computer science.

Among the many ways probabilistic choice can be captured in programming, the simplest one consists in endowing the language of programs with an operator modelling sampling from (one or many) distributions. Fair, binary, probabilistic choice is for example perfectly sufficient to get universality if the underlying programming language is itself universal [35, 8]. This is precisely what happened in the realm of the λ-calculus, from the pioneering works by Saheb-Djahromi dating back to the seventies [34] to the most recent contributions (e.g., [9, 14, 29, 21]), through the seminal work, e.g., of Jones and Plotkin [23].

Termination is a crucial property of programs, and remains desirable in a probabilistic setting, e.g., in probabilistic programming [18] where inference algorithms often rely on the underlying program to terminate. However, one needs first of all to understand *what it means* for a probabilistic computation to terminate, i.e., how termination should be defined in the first place. If one wants to stick to a *qualitative* definition, almost-sure termination is a well-known answer: a probabilistic computation is said to almost-surely terminate if divergence, although possible, has null probability. One could even go beyond and require *positive* almost-sure termination, which asks the average time to termination to be finite. This is well-known to be stronger than almost sure termination. Recursion-theoretically, checking (positive) almost-sure termination is harder than checking termination in deterministic computation, where termination is at least recursively enumerable, although undecidable: in a universal probabilistic imperative programming language,

*This work has been partially supported by ANR projects 14CE250005 ELICA, and 16CE250011 REPAS.

†LIPN, Université Paris-Nord, flavien.breuvart@lipn.univ-paris13.fr

‡University of Bologna & INRIA Sophia Antipolis, ugo.dallago@unibo.it

almost sure termination is Π^0_2 complete, while positive almost-sure termination is Σ^0_2 complete [25].

In the λ-calculus, termination is one of the best-studied verification problems [22, 24], and intersection types are well-known to be able not only to guarantee but also to *characterise* various notions of termination for λ-terms, including weak and strong normalisation [6, 2, 27]. Can all this be generalised to probabilistic λ-calculi? This paper gives a first complete, positive, answer to this question. This is however bound to be challenging: due to the aforementioned recursion-theoretic limitations, we cannot hope to get a system of intersection types in which almost-sure termination is witnessed by the existence of *one* type derivation, even if we are prepared—as in the deterministic setting—to drop decidability of type inference. If we could find one, that would contradict the Π^0_2 completeness of the underlying verification problem, since type derivations can, at least, be enumerated. In other words, the price of being higher in the arithmetical hierarchy needs to be paid, somehow.

In this paper, we attack this difficult problem following two unusual routes. We design a type system by observing that binary probabilistic choice can be seen as a form of reading operation: a read-only, use-once, Boolean is taken in input from the memory and execution proceeds dependently on the read value. This suggests a way to define a system of intersection types, which we call *oracle* types. The second route we follow consists in lifting types to distributions, thus switching to a *monadic* form of type. In both cases, and again due to the aforementioned recursion-theoretic limitations, the probability of convergence cannot be read from a *single* derivation, but from *countably many of them*. While for oracle types the probability of convergence is obtained as the *sum* of the weights of *all* type derivations, monadic types allow for the weight of single type derivations to *converge* to the target probability. As a consequence, they are arguably better tailored as a verification methodology, and we will indeed define a decidable approximation of them.

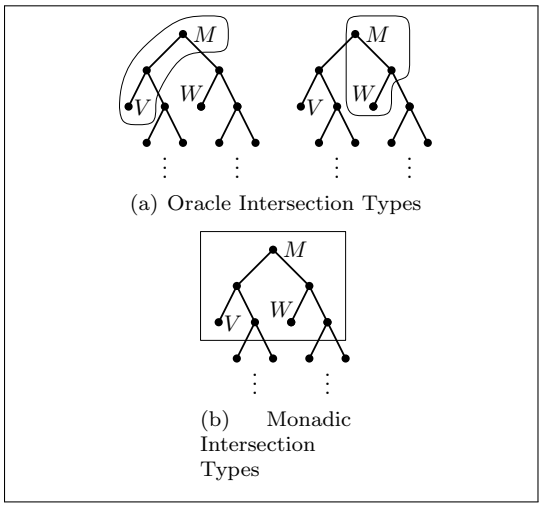
After presenting the termination problem informally (Section 2) and giving the necessary preliminaries about distributions and probabilistic λ-calculi (Section 3), this paper unfolds as follows:

- First, a system of *oracle* intersection types is introduced. The syntax of oracle intersection types explicitly mentions the bits the typed term is supposed to read from the oracle. In other words, choices performed along the execution are "hard-wired" into types, and each type derivation by construction talks about *one* probabilistic branch. One then only has to sum the *weights* of derivations representing different choice sequences. Soundness and completeness of typability with respect to the probability of convergence are proved by reducibility, and by subject *expansion*. This is the topic of Section 5.
- The second type system we present is motivated by termination as a verification problem, and can be seen as being designed from *prevision spaces* [19]. Probability distributions of types become first class objects, allowing for a new completeness theorem: the probability of termination of a term is *the least upper bound* of the norms of all type derivations for it. Moreover, with verification in mind, we refine our type system to suppress redundancies, and we show that there is at least one interesting fragment of it for which type inference is decidable. All this is in Section 6.

The differences between the two proposed type systems can be depicted as in Figure 1. The evaluation of a term M can be seen as being modeled as a (possibly infinite) tree whose leaves are values. While each oracle intersection type derivation captures *one* (finite) path leading M to a value (see Figure 1(a)), a monadic intersection type derivation captures *a finite portion* of the tree rooted at M obtained by pruning some (possibly infinite) subtrees (see Figure 1(b)).

2 On Types and Termination in a Probabilistic Setting

The untyped λ-calculus [3] is a minimalist formal system enjoying some remarkable properties like confluence and standardisation, which make it an ideal candidate for a reference model of pure functional programming. Key properties of terms, like being strongly or weakly normalisable, can be characterised (somehow "compositionally") by way of intersection-type disciplines [6, 2]. This



(a) Oracle Intersection Types

(b) Monadic Intersection Types

Figure 1: Intersection Types and Probabilistic Evolution.

is possible not only because normalisation has a semantic counterpart which can be captured by way of types but also, more fundamentally, because being normalisable is a recursively enumerable property: otherwise, one could not hope to get a type-based characterisation of it, unless type derivation checking (which is easier than type inference) turns out to be undecidable.

We can intuitively describe an intersection type derivation as a faithful, compositional, but "optimised" description of the evaluation of the underlying typed term. As an example, consider the term $M = (\lambda x.xx)\mathbf{I}$, where $\mathbf{I}$ is the identity $\lambda y.y$. It is clear that the variable x cannot be given just *one* simple type in the term above, and intersections are there precisely to account for the multiple uses *the same* subterm can be subject to. In the end, functions are not considered in their entirety but only on a finite number of arguments, namely those they will be fed with. In the example above, $\mathbf{I}$ needs to get *both* a type in the form $\beta = \alpha \to \alpha$ *and* the type $\beta \to \beta$, and M would thus have type β. The power of intersection types comes from the fact that any type derivation built according to them can be seen as a witness to termination, and that no information is lost this way.

In a probabilistic λ-calculus, this simple and beautiful picture is simply not there anymore. First of all, confluence does *not* hold, and this is not the mere consequence of the presence of probabilistic choice: it fails even if *all probabilistic execution branches* are taken into account, i.e. if one works with *distributions*. Consider, as an example, the term[1] $(\lambda x.x(x\mathbf{0}))\ (\boldsymbol{S} \oplus \boldsymbol{I})$, where $\oplus$ is an operator for fair, binary, probabilistic choice. When evaluated call-by-value (CBV for short), this term reduces to $\mathbf{0}$ with probability $\frac{1}{2}$ or to $\mathbf{2}$ with probability $\frac{1}{2}$. In call-by-name order (CBN for short), however, it reduces to $\mathbf{0}$ with probability $\frac{1}{4}$, to $\mathbf{2}$ with probability $\frac{1}{4}$ or to $\mathbf{1}$ with probability $\frac{1}{2}$. Simply put, duplication and probabilistic sampling do not commute. One avoid those issues by considering a fixed reduction strategy, which in this section will be, indeed, CBN. All the results we will give in this paper, however, hold both for CBN and CBV reduction. We will be keen to present both formulations, in order to highlight the (sometime subtle) differences.

The presence of probabilistic choice, together with the necessity of precisely capturing the behaviour of terms—we are aiming at completeness, after all—pose other challenges. Take, as an example, the term $(\lambda x.\mathbf{A} \oplus x)\ (\mathbf{B} \oplus \mathbf{C})$, where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are closed terms of types α, β and γ respectively. This term call-by-name reduces to $\mathbf{A}$ with probability $\frac{1}{2}$, to $\mathbf{B}$ with probability $\frac{1}{4}$ and to $\mathbf{C}$ with probability $\frac{1}{4}$. Thus, naively, we would like this term to somehow have the type $\frac{1}{2}\alpha + \frac{1}{4}\beta + \frac{1}{4}\gamma$. This requires types to include distributions on the left of the arrow, but this is not enough. Since typing needs to be somehow compositional, one would like to type separately $(\lambda x.\mathbf{A} \oplus x)$ and $(\mathbf{B} \oplus \mathbf{C})$, and the former has to be typed *without* knowing its arguments; ideally,

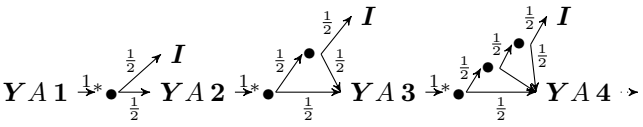
[1] where $\mathbf{0}$ is a numeral for 0, and $\boldsymbol{S}$ is the successor function.

it would be typed without even knowing *the type* of its arguments, but this is known to be incompatible with completeness [1]. Summing up, the language of types and typing rules are bound to be quite complex.

In addition to keeping track of the probability of certain events, we also have to deal with the multiple uses of the same variable by a term. E.g., in a term such as $\lambda x.xx$ that uses its argument twice and very differently, we need to require the variable x to have *two* different types, say α and β. Such an intersection $\alpha \wedge \beta$ thus means that the argument have to be of type α and of type β at the same time. Somehow, this issue is orthogonal to the one discussed above, and will thus have to be captured separately. Dealing with both of them *compositionally* is the main challenge we face, technically speaking.

As a slightly more complicated example, consider the term $(\lambda x.x\,(x\,x))\ (\lambda y.y \oplus \Omega)$. The three occurrences of x have very different roles: the rightmost one is only used as an argument, and there is no need for it to have a functional type; the one in the middle must have a functional type that takes a value (obtained with probability one) and returns something with probability $\frac{1}{2}$; the leftmost one takes an argument that may or may not converge, which means that the way it uses its argument matters a lot in the resulting probability. As a result, the management of the probabilistic behaviours we were talking about is different for each of the three occurrences of x.

Finally, it is worthwhile to notice that even if one forgets the technicalities related to the underlying type system, termination becomes itself an intrinsically more complex verification problem: a term (or, more generally, a probabilistic computation) does not merely *converge* to a normal form, but it does so *with a certain probability*. Moreover, the probability of convergence of a term is infinitary in nature. Let us consider, as an example, a term A such as $\lambda ux.x\,(\lambda y.(u\,(\boldsymbol{S}x) \oplus y))\,I$, where $\boldsymbol{S}$ is a combinator computing the successor on natural numbers. Let $\boldsymbol{Y}$ be Curry's fixpoint combinator. Then, the probabilistic evolution of the program $\boldsymbol{YA1}$ can be described by the following *infinite* tree:



In other words, $\boldsymbol{YA1}$ converges with probability 1, but this is witnessed by infinitely many (finite) probabilistic branches. Indeed, the already mentioned well-known results on the difficulty of verifying termination for probabilistic computations [25] can be rephrased as follows in any universal model of probabilistic computation like the untyped probabilistic λ-calculus [8] (and for any computable $0 < p \leq 1$):

- the lower bound problem "$\texttt{Prob}(M\Downarrow) > p$" is Σ^0_1-complete, *i.e.*, non decidable but recursively enumerable,
- the upper bound problem "$\texttt{Prob}(M\Downarrow) < p$" is Σ^0_2-complete, thus not recursively enumerable,
- the exact bound problem "$\texttt{Prob}(M\Downarrow) = p$" is Π^0_2 complete, thus not recursively enumerable.

All this implies, in particular, that *it is not possible* to give a type system where the exact (or even a upper) bound for the probability of termination of a term M is always proven by a finite derivation, unless checking the correctness of a derivation becomes undecidable.

Summing up, someone looking for a complete (intersection) type system for probabilistic λ-calculi would face two major challenges:

1. The first is, as we have seen, a severe constraint from recursion theory: almost sure termination is not a recursively enumerable property, and thus cannot be captured by a simple, recursively enumerable type system.
2. The second comes from the intrinsic complexity of dealing with probabilistic effects in a compositional way, and of keeping track of the dependencies between probabilistic choices, which themselves forces the type system to be complex, formally.

The first difficulty will be overcome by considering only the lower bound problem (which is recursively enumerable): a derivation computes a lower bound to the probability of termination, and the full probability of termination is obtained by considering *all* possible derivations. In other words, we will initially consider completely independent derivations corresponding to distinct possible executions of the term. We still have to be sure that distinct probabilistic branches correspond

$$\dfrac{\dfrac{\dfrac{\vdash \boldsymbol{A} : \alpha \qquad \overline{x : \left\langle \begin{smallmatrix} \beta \mapsto \frac{1}{2} \\ \gamma \mapsto \frac{1}{2} \end{smallmatrix} \right\rangle \vdash x : \left\langle \begin{smallmatrix} \beta \mapsto \frac{1}{2} \\ \gamma \mapsto \frac{1}{2} \end{smallmatrix} \right\rangle}}{x : \left\langle \begin{smallmatrix} \beta \mapsto \frac{1}{2} \\ \gamma \mapsto \frac{1}{2} \end{smallmatrix} \right\rangle \vdash \boldsymbol{A} \oplus x : \left\langle \begin{smallmatrix} \alpha \mapsto \frac{1}{2} \\ \beta \mapsto \frac{1}{4} \\ \gamma \mapsto \frac{1}{4} \end{smallmatrix} \right\rangle}}{\vdash \lambda x.\boldsymbol{A} \oplus x : \left\langle \begin{smallmatrix} \beta \mapsto \frac{1}{2} \\ \gamma \mapsto \frac{1}{2} \end{smallmatrix} \right\rangle \to \left\langle \begin{smallmatrix} \alpha \mapsto \frac{1}{2} \\ \beta \mapsto \frac{1}{4} \\ \gamma \mapsto \frac{1}{4} \end{smallmatrix} \right\rangle} \qquad \dfrac{\vdash \boldsymbol{B} : \beta \qquad \vdash \boldsymbol{C} : \gamma}{\vdash \boldsymbol{B} \oplus \boldsymbol{C} : \left\langle \begin{smallmatrix} \beta \mapsto \frac{1}{2} \\ \gamma \mapsto \frac{1}{2} \end{smallmatrix} \right\rangle}}{\vdash (\lambda x.\boldsymbol{A} \oplus x)\ (\boldsymbol{B} \oplus \boldsymbol{C}) : \left\langle \begin{smallmatrix} \alpha \mapsto \frac{1}{2} \\ \beta \mapsto \frac{1}{4} \\ \gamma \mapsto \frac{1}{4} \end{smallmatrix} \right\rangle}$$

(a) A CBN Type Derivation, where $\boldsymbol{A} : \alpha$, $\boldsymbol{B} : \beta$ and $\boldsymbol{C} : \gamma$.

$$\dfrac{\dfrac{\overline{x : \frac{1}{2}\mathcal{B} \to \frac{1}{4}\mathcal{B} \vdash x : \frac{1}{2}\mathcal{B} \to \frac{1}{4}\mathcal{B}} \qquad \dfrac{\overline{x : \mathcal{B} \to \frac{1}{2}\mathcal{B} \vdash x : \mathcal{B} \to \frac{1}{2}\mathcal{B}} \qquad \overline{x : \mathcal{B} \vdash x : \mathcal{B}}}{x : \left\{ \begin{smallmatrix} \mathcal{B} \\ \mathcal{B} \to \frac{1}{2}\mathcal{B} \end{smallmatrix} \right\} \vdash x\ x : \frac{1}{2}\mathcal{B}}}{x : \left\{ \begin{smallmatrix} \mathcal{B} \\ \mathcal{B} \to \frac{1}{2}\mathcal{B} \\ \frac{1}{2}\mathcal{B} \to \frac{1}{4}\mathcal{B} \end{smallmatrix} \right\} \vdash x\ (x\ x) : \frac{1}{4}\mathcal{B}}}{\vdash \lambda x.x\ (x\ x) : \left\{ \begin{smallmatrix} \mathcal{B} \\ \mathcal{B} \to \frac{1}{2}\mathcal{B} \\ \frac{1}{2}\mathcal{B} \to \frac{1}{4}\mathcal{B} \end{smallmatrix} \right\} \to \frac{1}{4}\mathcal{B}} \qquad \forall \mathcal{C},\ \vdash \lambda y.y \oplus \Omega : \mathcal{C} \to \frac{1}{2}\mathcal{C}}{\vdash (\lambda x.x\ (x\ x))\ (\lambda y.y \oplus \Omega) : \frac{1}{4}\mathcal{B}}$$

(b) A CBN Type Derivation Using Intersections.

Figure 2: Two Examples of Type Derivations.

to type derivations which can somehow be themselves told apart. Moreover, the complex internal dependencies will be flatten so that the term $(\lambda x.\boldsymbol{C} \oplus x)\ (\boldsymbol{D} \oplus \boldsymbol{E})$ will get four different derivations, which can be distinguished by the derived typing judgements, one for each of the 2^2 binary strings of length 2.

Unfortunately, the aforementioned type system is not well suited as a verification methodology. Indeed, we would like to trade completeness for a Σ_1^0 and thus approximable type-checking and type-inference without too much loss. However, a type derivation in the aforementioned system only gives information on *one* possible execution. In particular, we would like a system where the completeness is not obtained by *adding* the weights of different derivations, but as the *least upper bound* of those. With this goal in mind, we define a monadic intersection type system, which will be given in Section 6 below. A judgement in this system assigns a *distribution* of types rather than a type and its contexts associate to each variable a *set* of type distributions. This time, the execution tree is not explored depth-first, but breadth-first. A derivation corresponds to a finite portion of the execution tree. In this system, Figure 2(a) is a derivation for $(\lambda x.\mathbf{A} \oplus x)\ (\mathbf{B} \oplus \mathbf{C})$. Figure 2(b) is a type derivation for $(\lambda x.x\ (x\ x))\ (\lambda y.y \oplus \Omega)$. There, we assume a distribution of type $\mathcal{A}$ such that $\mathcal{B}$ is the Dirac distribution for $\mathcal{A} \to \frac{1}{2}\mathcal{A}$, so that the last deduction comes from the fact that $\mathcal{C}$ can be either $\mathcal{A}$, $\mathcal{B}$, or $\frac{1}{2}\mathcal{B}$. Notice the different "style" of those two derivations.

3 A Probabilistic λ-Calculus and Its Operational Semantics

3.1 Preliminaries

Let S be any countable set. We indicate the powerset of S as $\mathfrak{P}(S)$, and its restriction to finite sets as $\mathfrak{P}_f(S)$. We often describe sets using braces $\{-\}$. Similarly, we indicate as $\mathfrak{M}(S)$ the set of multisets of S, and as $\mathfrak{M}_f(S)$ its restriction to finite multisets. To distinguish multisets from sets, we describe multisets using square brackets $[-]$. Both finite sets and finite multisets are denoted with metavariables ranging over the lowercase alphabet letters $a, b, c, \ldots$.

We denote $\mathfrak{D}(S)$ the set of *probabilistic (sub)distributions over* S:

$$\mathfrak{D}(S) := \left\{ \mathcal{M} : S \to \mathbb{R}_{[0,1]} \;\middle|\; \sum_{M \in S} \mathcal{M}(M) \leq 1 \right\}.$$

Probabilistic distributions[2] form a ω**CPO** when endowed with the pointwise order. We write $SUPP(\mathcal{M})$ for the *support* of the distribution over $\mathcal{M}$, namely the set $\{M \in S \mid \mathcal{M}(M) > 0\}$. In particular $\mathfrak{D}_f(S)$ indicates the set of finitely supported distributions over S. Distributions are denoted with metavariable ranging over mathcal alphabet: $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$

Distributions are often indicated in square bracket notation. In most cases, we use the $\mapsto$ arrow to represent the non-null coefficient, e.g., $\left\langle \begin{smallmatrix} M \mapsto \frac{1}{2} \\ N \mapsto \frac{1}{2} \end{smallmatrix} \right\rangle$ is the uniform distribution over the 2-element set $\{M, N\}$, which will also be denoted as $\langle \frac{1}{2}M, \frac{1}{2}N \rangle$. We sometime indicate the Dirac distribution $\langle M \mapsto 1 \rangle$ as $\langle M \rangle$, while the *null* distribution is denoted $\langle\rangle$.

The set of functions (on the same set) whose codomain is the field of real numbers can be given themselves the status of a vector space. As a consequence, given two distributions $\mathcal{M}$ and $\mathcal{N}$ and a real number $p \leq 1$, we can indeed define the following:

$$\begin{aligned} \mathcal{M} + \mathcal{N} &:= \langle M \mapsto \mathcal{M}(M) + \mathcal{N}(M) \mid \\ &\qquad M \in SUPP(\mathcal{M}) \cup SUPP(\mathcal{N}) \rangle \ , \\ p\mathcal{M} &:= \langle M \mapsto p \cdot \mathcal{M}(M) \mid M \in SUPP(\mathcal{M}) \rangle \ . \end{aligned}$$

Observe, however, that $\mathcal{M} + \mathcal{N}$ is not necessarily a *distribution*: its sum can in general be greater than 1.

3.2 Probabilistic Abstract Reduction Systems

A *(fully) probabilistic abstract reduction system* (a *PARS*, for short) on S is a partial function from S to $\mathfrak{D}(S)$. Given a PARS $\to$, the fact that $(M, \mathcal{M}) \in \to$ is, as usual, indicated with $M \to \mathcal{M}$, and any $V \in S$ that cannot be reduced (i.e., $\to$ is undefined on V) is called *irreducible* or a *normal form*. The set of normal forms is denoted as S_V, while S_R is $S - S_V$. We also define the *reducible support* and *irreducible support* of $\mathcal{M} \in \mathfrak{D}(S)$ as $SUPP_R(\mathcal{M}) := SUPP(\mathcal{M}) \cap S_R$ and $SUPP_V(\mathcal{M}) := SUPP(\mathcal{M}) \cap S_V$, respectively. Any distribution $\mathcal{M}$ with empty reducible support is called itself *irreducible* and is indicated with metavariables like $\mathcal{V}$ or $\mathcal{W}$. $\mathcal{M}_V$ stands for the restriction of $\mathcal{M}$ to elements in S_V. Similarly for $\mathcal{M}_R$.

The function $\to$ can be generalised into a binary relation on $\mathfrak{D}(S)$ by taking it as the smallest such relation closed under the following rule:

$$\frac{\forall M \in SUPP_R(\mathcal{M}),\ M \to \mathcal{N}_M \qquad \forall V \in SUPP_V(\mathcal{M}),\ \mathcal{N}_M := \langle V \rangle}{\mathcal{M} \to \sum_M \mathcal{M}(M) \cdot \mathcal{N}_M} \ (r\text{-}\in)$$

Proposition 3.1 *If* $\mathcal{M} \to \mathcal{N}$ *then* $\mathcal{M}_V \leq \mathcal{N}_V$.

[2] In the following "distribution" always stands for "subdistribution".

Actually, $\rightarrow$ as we have just defined it is a (total) function. For $n \in \mathbb{N}$, we also define the n-step reduction $\rightarrow^n$ as the n-th iteration of $\rightarrow$: $\rightarrow^0$ is the identity on $\mathfrak{D}(T)$, while $\rightarrow^{n+1} := \rightarrow^n \circ \rightarrow$ for every $n \in \mathbb{N}$. The reflexive transitive closure $\bigcup_n \rightarrow^n$ of $\rightarrow$ is, as usual, indicated as $\rightarrow^*$. Please observe that $\rightarrow^n$ is again a total function for every $n \in \mathbb{N}$. For any term M and for every $n \in \mathbb{N}$, we can thus define $\mathcal{M}_n$ to be the unique distribution such that $M \rightarrow^n \mathcal{M}_n$. Due to the increasing character of $(\mathcal{M}_n)_V$, and to the fact that distributions form an $\omega\mathbf{CPO}$, we can define:

$$[\![M]\!] := \mathtt{sup}_n (\mathcal{M}_n)_V.$$

The distribution $[\![M]\!]$ is thus the distribution to which M converges *at the limit*. As such, it can be seen as the result of evaluating M.

3.3 Terms and Their Operational Semantics

The calculus we are interested in here, called $\Lambda_\oplus$, can be seen as being obtained by endowing the pure, untyped, λ-calculus with an operator for binary probabilistic choice. This makes it a PARS, which will be the subject of this section. To be precise, this will be done in *two* different ways, corresponding to two notions of evaluation for λ-terms: call-by-name reduction and call-by-value reduction.

The set of *terms* is defined as follows:

$$\Lambda_\oplus : \qquad L, M, N ::= x \;\Big|\; \lambda x.M \;\Big|\; M\ N \;\Big|\; M \oplus N.$$

Terms are taken modulo α-equivalence, which allows the definition of $M[N/x]$, the capture avoiding substitution of the term N for all free occurrences of x in M. The set of *closed terms* (i.e., terms in which no variable occurs free) is denoted as $\Lambda^C_\oplus$. A *term distribution* (a *closed term distribution*, respectively) is a probabilistic subdistribution over terms (over closed terms, respectively), i.e. an element of $\Lambda^{\mathcal{D}}_\oplus = \mathfrak{D}(\Lambda_\oplus)$ (of $\Lambda^{\mathcal{D}C}_\oplus = \mathfrak{D}(\Lambda^C_\oplus)$, respectively), which is ranged over by metavariables like $\mathcal{L}$ and $\mathcal{N}$. Whenever this does not cause ambiguity, we generalise syntactic operators to operators on (closed) term distributions, e.g. $M\mathcal{N}$ stands for the distribution assigning probability $\mathcal{N}(L)$ to ML.

As already mentioned, the operational semantics takes the form of a probabilistic abstract reduction system over terms.

Definition 3.2 *Rules for call-by-name (a.k.a. weak-head) and call-by-value reduction are given in Figures 3(a) and 3(b), respectively. This defines two PARSs $\rightarrow_N$ and $\rightarrow_V$ on $\Lambda^C_\oplus$.*

A quick inspection at the rules of Figure 3(a) and Figure3(b) reveals that (closed) normal forms of $\rightarrow_N$ and $\rightarrow_V$ are *closed values*, i.e. closed abstractions. Moreover, $\rightarrow_N$ and $\rightarrow_V$ are indeed partial functions as required by the definition of a PARS: all the rules are syntax-directed.

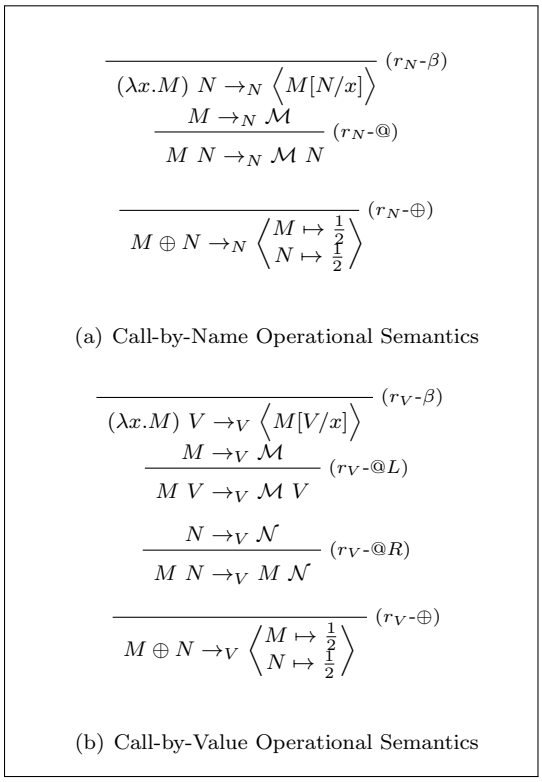
Following the development from Section 3.2 above, we can define the semantics (or evaluation) $[\![M]\!]_N$ and $[\![M]\!]_V$ as distributions over closed, irreducible terms. The *call-by-name probability of convergence* of any term M, then, is $\sum [\![M]\!]_N := \sum_V [\![M]\!]_N (V)$. Similarly for $\sum [\![M]\!]_V := \sum_V [\![M]\!]_V (V)$, the *call-by-value probability of convergence*. These notions of evaluation satisfy some interesting properties, such as the following continuity lemma:

Lemma 3.3 *For any pair of terms M, N, the following hold:*

$$[\![[\![M]\!]_N\ \ N]\!] = [\![M\ N]\!]_N\ , \qquad [\![[\![M]\!]_V\ \ [\![N]\!]_V]\!] = [\![M\ N]\!]_V\ .$$

4 Intersection Types: A Naïve Attempt

Let us start with two remarks. First, our type system should subsume an existing intersection type system on terms not containing the binary choice operator $\oplus$. Secondly, handling duplication requires some care, since duplicating an $\oplus$ operator *before* or *after* its evaluation can be drastically



(a) Call-by-Name Operational Semantics

(b) Call-by-Value Operational Semantics

Figure 3: Operational Semantics.

different (see the discussion on Section 2). It is thus natural to try to extend De Carvalho's non-idempotent intersection type system [11], which is known for being resource aware. In doing so, let us consider CBN evaluation.

We present a naïve intersection type system which is simply a decoration of De Carvalho's system with real numbers. A judgment $\Gamma \vdash M : p{\cdot}\alpha$ should be read as "there is an execution of the term M converging into a normal form of type α and this execution has probability p to occur".

The *naïve intersection types* for the probabilistic lambda calculus are either a *base type* (or *type variable*) $\star$, or the *arrow type* $a{\to}p{\cdot}\alpha$, where α is itself an intersection type, $p \in [0,1]$ is the weight of the function and a is a finite multiset of intersection types:

(Weights)		p	$\in$	$[0,1]$
(Types)	$\mathbb{T}_E$	$\alpha, \beta ...$	$:=$	$\star \mid a{\to}p{\cdot}\alpha$
(Intersections)	$\mathfrak{M}_f(\mathbb{T}_E)$	$a, b ...$	$:=$	$[\alpha_1, \ldots, \alpha_n]$

An *environment* $\Gamma : \mathbb{V} \to \mathfrak{M}_f(\mathbb{T}_E)$ is a function from variables to finite multisets of types. We define a (commutative) sum of contexts as the pointwise sum $\Gamma + \Delta := (x \mapsto \Gamma(x) \uplus \Delta(x))$. Moreover we use the following syntactic sugar: $(x : a)$ is the context that associates a to x and $[]$ to every other variable, while $(\Gamma; x : a) := \Gamma + (x : a)$ is defined only when $\Gamma(x) = []$. A *judgment* is of the form $\Gamma \vdash M : p{\cdot}\alpha$ for $p \in \mathbb{R}_{[0,1]}$, Γ a context, M a term and α a type. The real number p is called the *weight* of the judgment. Typing rules are given in Figure 4.

The rules (N-x) and (N-λ) are similar to De Carvalho's system, with a trivial weight 1. Notice that, since we are considering a weak notion of reduction, a λ-abstraction is irreducible, thus having maximal probability of normalisation. We need, however, to keep track of the probability of convergence of the content of those λ-abstractions *once applied to an argument*: this is the purpose of the weight p in the arrow type $a{\to}p{\cdot}\alpha$. The rules (N-$\oplus L$) and (N-$\oplus R$) are very natural probabilistic adaptations of the classical rules used for the non-deterministic operators.

$$\frac{}{x : [\alpha] \vdash x : 1\cdot\alpha}\,(N\text{-}x) \qquad \frac{}{\Gamma \vdash \lambda x.M : \star}\,(N\text{-}\star)$$

$$\frac{\Gamma \vdash M : p\cdot\alpha}{\Gamma \vdash M \oplus N : \frac{p}{2}\cdot\alpha}\,(N\text{-}\oplus L)$$

$$\frac{\Gamma; x : a \vdash M : p\cdot\alpha}{\Gamma \vdash \lambda x.M : 1\cdot(a \to p\cdot\alpha)}\,(N\text{-}\lambda)$$

$$\frac{\Gamma \vdash N : p\cdot\alpha}{\Gamma \vdash M \oplus N : \frac{p}{2}\cdot\alpha}\,(N\text{-}\oplus R)$$

$$\frac{\Gamma \vdash M : p\cdot([\alpha_1, ..., \alpha_n] \to q\cdot\beta) \qquad [\Delta_i \vdash N : r_i\cdot\alpha_i]_{1\le i\le n}}{\Gamma + \sum_i \Delta_i \vdash M\ N : p\cdot q\cdot(\prod_i r_i)\cdot\beta}\,(N\text{-}@)$$

Figure 4: The Naïve Intersection Type System.

The rule (N-@) seems complex, but it corresponds to the rule of De Carvalho's system with the intuitive probabilistic annotations.

Our hope, at this point, is to prove that derivability of $\vdash M : p\cdot\alpha$ and $\sum [\![M]\!] = p$ are equivalent. Unfortunately, this is *not* the case, and for very good reasons which have to do with the recursion-theoretic difficulty of proving a probabilistic computation to be, e.g., almost surely terminating. Indeed, the set of type derivations proving the judgement above is recursively enumerable, so the "existence of a type derivation of some $\vdash M : p\cdot\alpha$" is a Σ^0_1 property, which is incompatible with the Π^0_2-completeness of the set of almost-surely terminating terms [25].

In fact, a derivation of $\vdash M : p\cdot\alpha$ means that there is *one* possible normalising execution of M which happens with probability p. If we want to fully characterise the probability of normalisation, we need to consider derivations for *each* of the potential normalising executions of M. The probability of convergence becomes the sum of the weights of each derivation. This is not as easy as one could imagine, as the following example shows.

Take the term $(\lambda x.x\ (x\ I))\ (I \oplus I)$ where I is the identity $\lambda y.y$. This term evaluates to $\langle I \rangle$. However, there are four possible normalising executions—each copy of $I \oplus I$ results in a probabilistic choice. This means that we are expecting four type derivations, each with a weight of $\frac{1}{4}$. But, two of them *cannot* be really distinguished. In order to describe them, let us introduce the following shortcuts:

$$\alpha := [\star] \to 1\cdot\star\,, \qquad \beta := [\alpha] \to 1\cdot\alpha\,, \qquad \tau := [\beta, \beta] \to 1\cdot\alpha\,,$$

and define a type derivation π as follows

$$\dfrac{\dfrac{\dfrac{}{x : [\beta] \vdash x : 1\cdot\beta} \qquad \dfrac{\dfrac{}{x : [\beta] \vdash x : 1\cdot\beta} \qquad \dfrac{\dfrac{}{y : [\star] \vdash y : 1\cdot\star}}{\vdash I : 1\cdot\alpha}}{x : [\beta] \vdash x\ I : 1\cdot\alpha}\,(N\text{-}@)}{x : [\beta] \vdash x\ (x\ I) : 1\cdot\alpha}\,(N\text{-}\lambda)}{\vdash \lambda x.x\ (x\ I) : 1\cdot\tau}\,(N\text{-}\lambda)$$

and another type derivation ρ as follows:

$$\dfrac{\dfrac{}{x : [\alpha] \vdash x : 1\cdot\alpha}\,(N\text{-}x)}{\vdash I : 1\cdot\beta}\,(N\text{-}\lambda)\cdot$$

The different derivations are the following ones, differing only by the uses of rules (N-$\oplus L$) and (N-$\oplus R$):

$$\dfrac{\dfrac{\pi}{\vdash \lambda x.x\ (x\ I) : \tau} \qquad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}\,(N\text{-}\oplus L) \qquad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}\,(N\text{-}\oplus L)}{\vdash (\lambda x.x\ (x\ I))\ (I \oplus I) : \frac{1}{4}\cdot\alpha}\,(N\text{-}@)$$

$$\dfrac{\dfrac{\pi}{\vdash \lambda x.x\ (x\ I) : \tau} \quad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}(N\text{-}\oplus R) \quad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}(N\text{-}\oplus R)}{\vdash (\lambda x.x\ (x\ I))\ (I \oplus I) : \frac{1}{4}\cdot\alpha}(N\text{-}@)$$

$$\dfrac{\dfrac{\pi}{\vdash \lambda x.x\ (x\ I) : \tau} \quad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}(N\text{-}\oplus L) \quad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}(N\text{-}\oplus R)}{\vdash (\lambda x.x\ (x\ I))\ (I \oplus I) : \frac{1}{4}\cdot\alpha}(N\text{-}@)$$

Essentially, we would like to get the following fourth derivation

$$\dfrac{\dfrac{\pi}{\vdash \lambda x.x\ (x\ I) : \tau} \quad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}(N\text{-}\oplus R) \quad \dfrac{\dfrac{\rho}{\vdash I : 1\cdot\beta}}{\vdash I \oplus I : \frac{1}{2}\cdot\beta}(N\text{-}\oplus L)}{\vdash (\lambda x.x\ (x\ I))\ (I \oplus I) : \frac{1}{4}\cdot\alpha}(N\text{-}@)$$

but this derivation is in fact the same as the former one (the order between the two rightmost assumptions of the applicative rule does not matter and *cannot* matter). The issue comes from τ having twice the same type, so that we cannot distinguish the two different uses.

The naïve approach would be to use ordered sequences in place of multisets, leading to non-commutative intersection types. In fact, intersection types are not able to distinguish any non-commutative effect. This is because they do not track sequentiality of the execution. That is why we rather opt for a solution preserving subject reduction where the *effect* of flipping a fair coin is kept track in the judgment, and thus in the type, as we will detail in the following section.

There is only *one* example of a probabilistic intersection type system in the literature [10, 14]. There, Ehrhard, Pagani and Tasson derive an intersection type system from probabilistic coherent spaces. This system was a source of inspiration for us, but is fundamentally different from what we propose in this paper, for two important reasons:

- They consider *head* reduction rather than *weak head* reduction. As a result, probabilities associated with higher-order terms hardly make any sense. Since they consider **PCF**, they bypass this issue by using first order observations, but in the untyped λ-calculus this is not possible.
- The issue of distinguishing between identical derivations is resolved in a rather crude way: each subderivation is annotated with its interpretation in coherent spaces, this way differentiating fundamentally different ones. We are looking for more resonable solutions to this issue.

5 Achieving Completeness: Oracle Intersection Types

How should we alter the naïve intersection type system we introduce in the last section, so as to make it capable of capturing distinct probabilistic evolutions of a term by distinct type derivations? Ultimately, what made our type system *not* capable of distinguishing two essentially different sequences of probabilistic choices is that only their *probability* is taken into account, while the outcome of the choice is simply discarded. In other words, there is nothing in the type of $M \oplus N$ telling us that M and all its reducts will be reached only if the outcome of the first probabilistic choice we perform is, say, 0 rather than 1: the only thing we remember is that this path(s) have a (combined) probability of at most $\frac{1}{2}$.

The basic idea behind oracle intersection types consists in substituting real numbers in naïve intersection types with strings, this way allowing to switch to a simple type system in which intersection becomes idempotent. This works both when call-by-name and call-by-value evaluation is considered, although the type systems needs to be tuned. This is what we are going to do in this section.

It is worth noticing that our approach, in the spirit, follows the ideas underlying Goubault-Larrecq and Varacca's "Continuous Random Variables" [20]. However, further study is needed to make this link formal.

$$\frac{s\cdot\alpha \in a}{\Gamma, x : a \vdash x : s\cdot\alpha}\ (I_N\text{-}x) \qquad \frac{\Gamma; x : a \vdash M : s\cdot\alpha}{\Gamma \vdash \lambda x.M : \varepsilon\cdot(a\to s\cdot\alpha)}\ (I_N\text{-}\lambda) \qquad \frac{}{\Gamma \vdash \lambda x.M : \varepsilon\cdot\star}\ (I_N\text{-}\star)$$

$$\frac{\Gamma \vdash M : s\cdot\alpha}{\Gamma \vdash M \oplus N : (0s)\cdot\alpha}\ (I_N\text{-}\oplus L)$$

$$\frac{\Gamma \vdash M : r\cdot(\{s_k\cdot\alpha_k\}_{k\in K}\to(t\cdot\beta)) \qquad \left\{\Gamma \vdash N : s_k\cdot\alpha_k\right\}_{k\in K}}{\Gamma \vdash M\ N : (rt)\cdot\beta}\ (I_N\text{-}@)$$

$$\frac{\Gamma \vdash N : s\cdot\alpha}{\Gamma \vdash M \oplus N : (1s)\cdot\alpha}\ (I_N\text{-}\oplus R)$$

Figure 5: The Intersection Type System $\mathbf{I}_N$.

5.1 System $\mathbf{I}_N$

Let us first describe a system of oracle intersection types fitted for the PARS $\to_N$. Since, in call-by-name, arguments are passed to functions *unevaluated*, we are bound to work with types in which intersections appear in negative positions, i.e., to the left of the arrow. But how about a function's *result*? Actually, an *oracle intersection type* is either the base type or an arrow type $a\to(s\cdot\alpha)$, where $(s\cdot\alpha)$ is a *weighted* intersection type, i.e., a pair of a string $s \in \{0,1\}^*$ and an intersection type α, and a is a set of weighted intersection types. Formally:

(Weights)	$\mathbb{B}$	s	$\in$	$\{0,1\}^*$
(Types)	$\mathbb{O}_N$	$\alpha,\beta...$	$:=$	$\star \mid a\to(s\cdot\alpha)$
(Intersections)	$\mathbb{O}_N^{(\mathbb{B})}$	$a,b...$	$:=$	$\{(s_1\cdot\alpha_1),\dots,(s_n\cdot\alpha_n)\}$

For example, a term like $\lambda x.x \oplus \Omega$ can be given the type $\{1\cdot\star\} \to 01\cdot\star$, meaning that on an input which evaluates to a value after following the right branch of a probabilistic choice (and thus having the type $1\cdot\star$), the function results in a term which evaluates to a value after performing *two* probabilistic choices, the first one taking the left branch, and the second one taking the right branch (and thus having the type $01\cdot\star$).

An *environment* $\Gamma : \mathbb{V} \to \mathbb{O}_N^{(\mathbb{B})}$ is a function from variables to intersections. We define a commutative sum of environments as $\Gamma \cup \Delta := (x \mapsto \Gamma(x) \cup \Delta(x))$. Moreover we use the following syntactic sugar: $(x : a)$ is the environment associating a to x and $\emptyset$ to every other variable, and $(\Gamma; x : a) := \Gamma \cup (x : a)$ is defined only whenever $\Gamma(x) = \emptyset$. A *sequent* is of the form $\Gamma \vdash M : s\cdot\alpha$, for $s \in \{0,1\}^*$, Γ an environment, M a term and α a type. The binary string s is called the *weight* of the sequent.

The intersection type system $\mathbf{I}_N$ is given in Figure 5. First of all, please observe how values can be typed in two different ways, namely by an arrow type and by $\star$. In both cases, the underlying binary string is ε, since values are irreducible and thus cannot be fired. Consider now rule $(I_N\text{-}@)$: the string rt in the conclusion is the concatenation of two strings in the function M, while the strings s_k in the arguments are simply not taken into account. This corresponds to the fact that arguments are passed to functions *unevaluated*.

The following is the analogue of the classic Subject Reduction Theorem:

Proposition 5.1 (Subject Reduction)
If $\vdash M : s\cdot\alpha$ *and* $M \to_N \langle N_i \mapsto p_i\rangle_{1\le i\le n}$ *then:*

- *Either* $n = 1$ *and* $\vdash N_1 : s\cdot\alpha$*;*
- *Or* $n = 2$*,* $s = br$*,* $p_1 = p_2 = 1/2$ *and there is* $i \in \{1,2\}$ *such that* $\vdash N_i : r\cdot\alpha$*.*

Given a string $s \in \{0,1\}^*$, *the probability of* s is naturally defined as $2^{-|s|}$. Given a set of binary strings $S \subseteq \{0,1\}^*$, the expression $2^{-|S|}$ stands for the sum $\sum_{s\in S} 2^{-|s|}$, also called *the probability of* S. Given a term M, the set $\mathcal{E}_N(M) \subseteq \{0,1\}$ of *binary strings for* M is defined as follows:

$$\mathcal{E}_N(M) := \{s \in \{0,1\}^* \mid \exists\alpha.\ \vdash M : s\cdot\alpha\}.$$

$$\frac{}{x : a \vdash x : \varepsilon \cdot a}\ (I_V\text{-}x) \qquad \frac{\left\{\Gamma_k; x : a_k \vdash M : s_k \cdot b_k \ \right\}_{k \in K}}{\cup_{k \in K} \Gamma_k \vdash \lambda x.M : \varepsilon \cdot \{a_k \to s_k \cdot b_k\}_{k \in K}}\ (I_V\text{-}\lambda)$$

$$\frac{\Gamma \vdash M : r \cdot [a \to t \cdot b] \qquad \Delta \vdash N : s \cdot a}{\Gamma \cup \Delta \vdash M\ N : (rst) \cdot b}\ (I_V\text{-@}) \qquad \frac{\Gamma \vdash M : s \cdot a}{\Gamma \vdash M \oplus N : (0s) \cdot a}\ (I_V\text{-}\oplus L)$$

$$\frac{\Gamma \vdash N : s \cdot a}{\Gamma \vdash M \oplus N : (1s) \cdot a}\ (I_V\text{-}\oplus R)$$

Figure 6: The Intersection Type System $\mathbf{I}_V$. In rule ($\mathbf{I}_V$-λ), a proof has to be given for each sequent of the set.

We are now in a position to state completeness of $\mathbf{I}_N$ as a verification methodology for almost-sure termination, and more generally as an inference methodology for the probability of termination:

Theorem 5.2 *Let M be any closed term. The probability of convergence of M is the sum of the probabilities of the binary strings for M:*

$$\sum [\![M]\!] \ = 2^{-|\mathcal{E}_N(M)|}.$$

Proof. We consider a λ-calculus with a linear read-only binary stream as a state, defined with a read operator $\boxplus$ such that $(0s, M \boxplus N) \to (s, M)$ and $(1s, M \boxplus N) \to (s, N)$. We consider a weak-head convergence to a term with an empty stream, so that $(01, \lambda x.x)$ is a diverging term. A reducibility argument, together with Subject Expansion, guarantees that (s, M) converges iff $\vdash M : s.\alpha$ for some α. What remains to be done, then, is to prove the following two implications:

- If $M \to_N^* \mathcal{M}$ then there is a set X of binary strings such that $\sum_{s \in X} 2^{-|s|} \geq \sum \mathcal{M}$ and for every $s \in X$ it holds that $(s, M^{\oplus 2 \boxplus})$ converges.
- Let X be any set of binary strings such that for every $s \in X$ it holds that (s, M) converges. Then there is a distribution such that $(M)^{\boxplus 2 \oplus} \to_N^* \mathcal{M}$ and $\sum \mathcal{M} \geq \sum_{s \in X} 2^{-|s|}$.

where $(\cdot)^{\oplus 2 \boxplus}$ and $(\cdot)^{\boxplus 2 \oplus}$ are simple translations that switches between the operators $\oplus$ and $\boxplus$. $\square$

5.2 System $\mathbf{I}_V$

Is there a way to fit $\mathbf{I}_N$ to call-by-*value* evaluation? That is a very relevant question, given that most effectful languages indeed adopt eager evaluation—otherwise there would be no way of re-using the outcome of probabilistic sampling. The system $\mathbf{I}_V$ can be seen as designed from Girard's "boring" translation [16, 30] in the same way $\mathbf{I}_N$ is designed from the standard encoding of intuitionistic logic. Apart from that, there is no other essential difference between the two systems, with the exception of string annotations, which are placed only at the right of the arrow in $\mathbf{I}_V$, following the lifting of monads in CBV.

Intersection types take here the form $a \to s \cdot b$, where $s \in \{0,1\}^*$ is the weight of the function and where both a and b are *sets* of intersection types. Formally:

$$\begin{array}{ccccc} \text{(Weights)} & & s & \in & \{0,1\}^* \\ \text{(Types)} & \mathbb{O}_V & \alpha, \beta ... & := & a \to s \cdot b \\ \text{(Intersections)} & \mathbb{O}_V^{(\mathbb{B})} & a, b ... & := & \{\alpha_1, \ldots, \alpha_n\} \end{array}$$

Environments are defined in the natural way, *judgments* are of the form $\Gamma \vdash M : s \cdot a$ for $s \in \{0,1\}^*$, Γ a context, M a term and a an intersection type.

The intersection type system $\mathbf{I}_V$ is given in Figure 6. Please observe how (closed) values continue to be annotated with the empty string. On the other hand, there is a striking difference in the way applications are treated: in CBN the probabilistic choices an application MN performs are those produced by M, followed by those produced by the λ-abstraction to which M evaluates

when applied to N. In CBV, terms are passed to functions *evaluated*, and this must be taken into account.

Given a term M, the set $\mathcal{E}_V(M) \subseteq \{0,1\}$ of *binary strings for* M is defined as follows:

$$\mathcal{E}_V(M) := \{s \in \{0,1\}^* \mid \exists a.\ \vdash M : s \cdot a\}.$$

This results in a theorem analogous to Theorem 5.2:

Theorem 5.3 *Let M be any closed term. The probability of convergence of M is the sum of the probabilities of the binary strings for M:*

$$\sum [\![M]\!] \ = \ 2^{-|\mathcal{E}_V(M)|}.$$

The proof of Theorem 5.3 is very similar, almost identical in structure, to the one of Theorem 5.2.

5.3 On Recursion Theory

Despite their simplicity, the type systems $\mathbf{I}_V$ and $\mathbf{I}_N$ are optimal, recursion theoretically. Indeed, consider the following formula

$$F(M) = \forall n \in \mathbb{N}. \exists X \subseteq_f \{0,1\}^*. \exists Y \subseteq_f \{0,1\}^*. P(M, n, X, Y)$$

where $P(M, n, X, Y)$ encodes the fact that for any $x \in X$ there (an encoding of) a type derivation $y \in Y$ whose conclusion is $\vdash M : x \cdot \alpha$ *and* $\sum_{x \in X} 2^{-|x|} \geq 1 - 2^{-n}$. Observe that the truth value of $P(M, n, X, Y)$ can be computed in elementary time. The fact that $F(M)$ exactly captures almost surely termination of M is a consequence of soundness and completeness. Finally, notice that F is a Π^0_2-formula.

6 Trading Elegance for Tractability: Monadic Intersection Types

Until now, we have seen that usual intersection type systems can indeed been refined to a degree of resource awareness allowing to track *all* the probabilistic choices performed along a computation, by embedding string annotations into types. The completeness of oracle intersection types is certainly of theoretical importance: it tells us that there is *a* way to adapt intersection type disciplines to probabilistic λ-calculi which is optimal recursion-theoretically. However, oracle intersection types are lacking as a practical verification methodology. And this is for several reasons:

- Having to look for several "independent" derivations before obtaining a sufficiently precise result is a computationally heavy process.
- There is no easy way to turn the type system into a methodology for inferring lower bounds on the probability of termination of a given term.

This section is devoted to introducing two type systems which go beyond oracle intersection types and towards a more tractable type system. However, the resulting system is bound to be complicated.

From a monadic point of view, being capable of computing type distributions means that we want to switch to a proper probabilistic monad $\mathfrak{D}$. However, $\mathfrak{D}$ is infamous for not being distributive with respect to the powerset comonad $\mathfrak{B}$. Much ongoing work has been devoted to trying to reconcile probabilistic distributions and powersets (e.g., [37]).

Fortunately, we will not have to deal with this issue directly. Indeed, in our systems, we do not need any distributive rule. The reason for that is that CBV and CBN are somehow symmetric: in CBV, we can use Girard "boring translation" [16] which does not fully use the comonadic part, while in CBN, we use a monadic version of this translation [31] as a degenerate way to treat monads in CBN.

$$\frac{\mathcal{A} \in a}{\Gamma; x : a \vdash x : \mathcal{A}}\ (MI_N\text{-}x) \qquad \frac{\Gamma \vdash M : \mathcal{A} \qquad \Gamma \vdash N : \mathcal{B}}{\Gamma \vdash M \oplus N : \frac{1}{2}\mathcal{A} + \frac{1}{2}\mathcal{B}}\ (MI_N\text{-}\oplus)$$

$$\frac{}{\Gamma \vdash M : \langle\rangle}\ (MI_N\text{-}\langle\rangle)$$

$$\frac{\Gamma; x : a \vdash M : \mathcal{A}}{\Gamma \vdash \lambda x.M : \langle a \to \mathcal{A}\rangle}\ (MI_N\text{-}\lambda)$$

$$\frac{\Gamma \vdash M : \mathcal{A} \qquad \left\{\Gamma \vdash N : \mathcal{B} \;\middle|\; \forall (a \to \mathcal{C}) \in SUPP(\mathcal{A}),\ \forall \mathcal{B} \in a\right\}}{\Gamma \vdash M\ N : \sum_{a \to \mathcal{C}} \mathcal{A}(a \to \mathcal{C}).\mathcal{C}}\ (MI_N\text{-}@)$$

Figure 7: The Monadic Intersection Type System $\mathbf{MI}_N$.

6.1 A New Paradigm: Monadic Intersection Types

Morally, what we have to do is to systematically superpose different oracle type derivations. This transformation alone, however, would not lead to a correct and complete type system. In fact, one more modification is needed: adopting sets *of distributions* over types rather that sets of types as intersections.

The *monadic intersection types* for the probabilistic lambda calculus are all *arrow types* $a \to \mathcal{A}$, where $\mathcal{A}$ is a distribution over intersection types with a finite support[3] and a is a set of distributions over intersection types. This means that we are using the informal identity:

$$\mathbb{M}_N \ \simeq\ \mathfrak{P}_f(\mathfrak{D}_f(\mathbb{M}_N)) \to \mathfrak{D}_f(\mathbb{M}_N)\ .$$

More formally, we are using the following grammar:

(Types)	$\mathbb{M}_N$	$\alpha, \beta, \gamma...$	$:=$	$a \to \mathcal{A}$
(Distributions)	$\mathfrak{D}_f(\mathbb{M}_N)$	$\mathcal{A}, \mathcal{B}, \mathcal{C}...$	$:=$	$\langle \alpha_i \mapsto p_i \rangle_{i \leq n}$
(Intersections)	$\mathfrak{P}_f(\mathfrak{D}_f(\mathbb{M}_N))$	$a, b, c...$	$:=$	$\{\mathcal{A}_1, \cdots, \mathcal{A}_n\}$

Notice that $\star := \emptyset \to \langle\rangle$ can be defined as syntactic sugar.

The *environments* $\Gamma : \mathbb{V} \to \mathfrak{P}_f(\mathfrak{D}_f(\mathbb{M}_N))$ are similar to those of the previously introduced intersection type systems. A *judgment* is of the form $\Gamma \vdash M : \mathcal{A}$ for Γ an environment, M a term and $\mathcal{A}$ a distribution over types.

The monadic intersection type system $\mathbf{MI}_N$ is given in Figure 7. Notice how the rule ($\mathbf{MI}_N$-$\oplus$) explores both probabilistic branches, while in oracle intersection types only one was considered in any type derivation. This allows us to get more refined derivations, but requires another rule, namely ($\mathbf{MI}_N$-$\langle\rangle$), to cut off infinite branches in the execution. This rule is very similar to the ω rule in usual intersection types, since the empty type plays the role of ω.

Theorem 6.1 (Subject Reduction/Expansion) *If $M \to \mathcal{N}$ then $\vdash M : \mathcal{A}$ iff $\vdash \mathcal{N} : \mathcal{A}$ where $\vdash \mathcal{N} : \mathcal{A}$ means that $(\vdash N : \mathcal{A}_N)_N$ for some decomposition $\mathcal{A} = \sum_N \mathcal{N}(N)\mathcal{A}_N$.*

The *weight* of a derivation π of $\Gamma \vdash M : \mathcal{A}$ is the norm $\sum \mathcal{A}$ of the type distribution. With such a definition, we get a correctness and completeness theorem that we claim more satisfactory than Theorem 5.2, as far as verification is concerned:

Theorem 6.2 *Let M be any closed term. The probability of CBN-convergence of M is the sup of the weights of its derivations:*

$$\sum [\![M]\!] \ =\ \bigvee_{\vdash M : \mathcal{A}} \sum \mathcal{A}$$

[3]Infinite distribution would make sense if one adds some special derivation for fixedpoint. But this would first require to formalise the inductive creation of types.

Proof. Soundness is proved by the usual reducibility argument. Reducibility candidates are not defined as sets but by the following relations

$$\begin{array}{lll} V \Vdash_V a{\to}\mathcal{A} & \text{iff} & \forall M \Vdash_S a,\ (V\ M) \Vdash_T \mathcal{A} \\ M \Vdash_S a & \text{iff} & \forall \mathcal{A} \in a,\ M \Vdash_T \mathcal{A} \\ \mathcal{V} \Vdash_D \mathcal{A} & \text{iff} & \mathcal{V} \widetilde{\Vdash_V} \mathcal{A} \\ M \Vdash_T \mathcal{A} & \text{iff} & [\![M]\!] \Vdash_D \mathcal{A} \\ \mathcal{M} \Vdash_T \mathcal{A} & \text{iff} & [\![\mathcal{M}]\!] \Vdash_D \mathcal{A} \end{array}$$

where $\widetilde{\Vdash_V}$ is the right-lax coupling relation over $\Vdash_V$ (see [4]), i.e., $\mathcal{V} \widetilde{\Vdash_V} \mathcal{A}$ if there is $\sigma \in \mathfrak{D}(\Vdash_V)$ such that $\mathcal{V}(V) = \sum_\alpha \sigma(V,\alpha)$ and $\mathcal{A}(\alpha) \leq \sum_V \sigma(V,\mathcal{A})$. The proof that $\vdash M{:}\mathcal{A}$ implies $M \Vdash_T \mathcal{A}$ follows the usual induction over the derivation of $\vdash M{:}\mathcal{A}$, using a saturation theorem; we only need to be careful while manipulating distributions. Completeness is proved by subject expansion as usual. □

Remark that the rule $(MI_N\text{-}\langle\rangle)$ can be replaced by a more expressive one allowing to combine different sub-derivations:

$$\frac{\left\{\Gamma \vdash M : \mathcal{A}_u \;\middle|\; u \in U\right\} \qquad \mathcal{U} \in \mathfrak{D}(U)}{\Gamma \vdash M : \sum_u \mathcal{U}(u)\mathcal{A}_u}\ (MI_N\text{-}D)$$

This rule is correct, but is not necessary to get completeness of $\mathbf{MI}_N$. However, it will turn out to be necessary for the completeness of the call-by-value version.

6.2 On Call-by-Value Evaluation

The same ideas we used to build $\mathbf{I}_V$ from $\mathbf{I}_N$ can be used to turn monadic intersection types into their call-by-value counterparts. However, monadic intersection types are notationally less elegant in CBV.

Our first compromise concerns the target calculus: for the sake of simplicity we will not consider the full calculus $\Lambda_\oplus$, but the sub-calculus containing only let-normal-forms:

Definition 6.3 *A* let-normal form *is a term of the form:*

$$M, N \ :=\ V \ \big|\ V\ M \qquad\qquad V, W \ :=\ x \ \big|\ \lambda x.M.$$

Any term can be turned into a let-normal form by eta-expending any subterm $(M\ N)$ into $(\lambda x.(\lambda y.y\ x)\ M)\ N$.

Another source of complexity comes from the type system itself: contrary to CBN, we have to add the convexity rule $(MI_V\text{-}D)$ to get completeness. Since we only need it when typing abstractions, we merge this rule into $(MI_V\text{-}\lambda)$. In addition, the rule for applications is more complex in order to preserve this convexity.

The *call by value monadic intersection types* are *arrow types* $a{\to}\mathcal{A}$, where a is a set of intersection type and $\mathcal{B}$ is a distribution over sets of intersection types. This means that we are using the informal identity:

$$\mathbb{M}_V \ \simeq\ \mathfrak{P}_f(\mathbb{M}_V) \to \mathfrak{D}_f(\mathfrak{P}_f(\mathbb{M}_V))\ .$$

More formally, we are using the following grammar:

$$\begin{array}{cccll} \text{(Types)} & \mathbb{M}_V & \alpha,\beta,\gamma... & := & a{\to}\mathcal{A} \\ \text{(Distributions)} & \mathfrak{D}_f(\mathfrak{P}_f(\mathbb{M}_V)) & \mathcal{A},\mathcal{B},\mathcal{C}... & := & \langle a_i \mapsto p_i\rangle_{i\leq n} \\ \text{(Intersections)} & \mathfrak{P}_f(\mathbb{M}_V) & a,b,c... & := & \{\alpha_1,\cdots,\alpha_n\} \end{array}$$

In addition, we define a stability condition on intersections that is not to be always respected but is only necessary in the left arguments of applications:

$$\frac{a \supseteq b}{\Gamma; x : a \vdash x : \langle b \rangle}\ (MI_V\text{-}x) \qquad \frac{\Gamma \vdash M : \mathcal{A} \qquad \Gamma \vdash N : \mathcal{B}}{\Gamma \vdash M \oplus N : \frac{1}{2}\mathcal{A} + \frac{1}{2}\mathcal{B}}\ (MI_V\text{-}\oplus)$$

$$\frac{\Gamma \vdash V : \langle a \rangle \qquad \Gamma \vdash N : \sum_{c \in \downarrow_s a} \mathcal{B}_c}{\Gamma \vdash V\ N : \sum_{b,c} \mathcal{B}_c(b)\hat{c}(b)}\ (MI_V\text{-}@)$$

$$\frac{\left\{\Gamma; x : a_i \vdash M : \mathcal{B}_{i,u} \;\middle|\; i \leq n, u \in U_i\right\} \qquad \forall i, \mathcal{U}_i \in \mathfrak{D}(U_i)}{\Gamma \vdash \lambda x.M : \left\langle \left\{ a_i \to \sum_u \mathcal{U}_i(u)\mathcal{B}_{i,u} \;\middle|\; i \leq n \right\} \right\rangle}\ (MI_V\text{-}\lambda)$$

Figure 8: The Monadic Intersection Type System $\mathbf{MI}_V$.

Definition 6.4 *A set $b \in \mathfrak{P}_f(\mathbb{M}_V)$ is* stable *if any two different arrows $(a_1 \to \mathcal{A}_1), (a_2 \to \mathcal{A}_2) \in b$, can be subsumed by a third one $((a_1 \cup a_2) \to \mathcal{B}) \in b$ such that $\mathcal{B} \geq \mathcal{A}_1, \mathcal{A}_2$ with the pointwise order. The interest of stable sets is that any stable a can be lifted into a function $\hat{a} : \mathfrak{P}_f(\mathbb{M}_V) \mapsto \mathfrak{D}_f(\mathfrak{P}_f(\mathbb{M}_V))$ by:*

$$\hat{a}(b) := \bigvee \{\mathcal{A} \mid \exists c \subseteq b, (c \to \mathcal{A}) \in a\}.$$

We also define, for all set a, the powerset of its stable subsets

$$\downarrow_s a := \{c \subseteq a \mid c \text{ is stable}\}$$

The *environments* $\Gamma : \mathbb{V} \to \mathfrak{P}_f(\mathbb{M}_V)$ are defined similarly to the ones for $\mathbf{MI}_N$. Notice that types of *values* and variables are necessarily Diracs. A *judgment* is of the form $\Gamma \vdash M : \mathcal{A}$ for Γ an environment, M a term and $\mathcal{A}$ distribution over types. The call-by-value monadic intersection type system $\mathbf{MI}_V$ is given in Figure 8.

Lemma 6.5 (Subject Reduction) *For any term M in let-normal form, if $M \to \mathcal{N}$ then $\vdash M : \mathcal{A}$ implies $\vdash \mathcal{N} : \mathcal{A}$ where $\vdash \mathcal{N} : \mathcal{A}$ means that $(\vdash N : \mathcal{A}_N)_N$ for some decomposition $\mathcal{A} = \sum_N \mathcal{N}(N)\mathcal{A}_N$.*

Remark that we do *not* claim Subject Expansion, as we did in $\mathbf{MI}_N$. In fact, we did prove the subject expension, but for a richer system whose other properties are equivalent. As for $\mathbf{MI}_N$, the *weight* of a $\mathbf{MI}_V$ derivation of the judgement $\Gamma \vdash M : \mathcal{A}$ is the norm $\sum \mathcal{A}$ of the type distribution.

Theorem 6.6 *Let M be any closed term in let-normal form. The probability of CbV-convergence of M is the least upper bound of the weights of its derivations:*

$$\sum [\![M]\!] \;=\; \bigvee_{\vdash M : \mathcal{A}} \sum \mathcal{A}$$

6.3 Trading Completeness for Decidability

Intersection type systems are well-known to have undecidable (but semi-decidable) type inference problems, due to their completeness. This also holds for all the systems we have introduced in this paper. A natural way to get a type system with decidable type inference out of an intersection type system is to get rid of intersections, thus collapsing down to a propositional type system. We also abolish general distribution types and only consider sub-Diracs, i.e., arrow types are of the form $\{p\langle\alpha\rangle\} \to q\langle\beta\rangle$, which we denote $p \cdot \alpha \to q \cdot \beta$.

We call these restricted types *probabilistic simple types*, and the corresponding type system $\mathbf{PST}_N$. Its rules are given in Figure 9(a). As a subsystem of $\mathbf{MI}_N$, system $\mathbf{PST}_N$ is sound, which is an easy consequence of the following theorem:

$$\frac{}{\Gamma; x : p{\cdot}\alpha \vdash x : p{\cdot}\alpha}\ (S_N\text{-}x) \qquad \frac{\Gamma \vdash M : p{\cdot}\alpha \qquad \Gamma \vdash N : q{\cdot}\alpha}{\Gamma \vdash M \oplus N : \frac{p+q}{2}{\cdot}\alpha}\ (S_N\text{-}\oplus)$$

$$\frac{}{\Gamma \vdash M : 0{\cdot}\alpha}\ (S_N\text{-}\langle\rangle)$$

$$\frac{\Gamma; x : p{\cdot}\alpha \vdash M : q{\cdot}\beta}{\Gamma \vdash \lambda x.M : 1{\cdot}(p{\cdot}\alpha{\to}q{\cdot}\beta)}\ (S_N\text{-}\lambda) \qquad \frac{\Gamma \vdash M : p{\cdot}(r{\cdot}\alpha{\to}q{\cdot}\beta) \qquad \Gamma \vdash N : r{\cdot}\alpha}{\Gamma \vdash M\ N : (pq){\cdot}\beta}\ (S_N\text{-}@)$$

(a) The Call-by-Name Probabilistic Simple Type System $\mathbf{PST}_N$

$$\frac{}{\Gamma; x : \alpha \vdash x : 1{\cdot}\alpha}\ (S_V\text{-}x) \qquad \frac{\Gamma \vdash M : p{\cdot}\alpha \qquad \Gamma \vdash N : q{\cdot}\alpha}{\Gamma \vdash M \oplus N : \frac{p+q}{2}{\cdot}\alpha}\ (S_V\text{-}\oplus)$$

$$\frac{}{\Gamma \vdash M : 0{\cdot}\alpha}\ (S_V\text{-}\langle\rangle)$$

$$\frac{\Gamma; x : \alpha \vdash M : p{\cdot}\beta}{\Gamma \vdash \lambda x.M : 1{\cdot}(\alpha{\to}p{\cdot}\beta)}\ (S_V\text{-}\lambda) \qquad \frac{\Gamma \vdash M : q{\cdot}(\alpha{\to}r{\cdot}\beta) \qquad \Gamma \vdash N : p{\cdot}\alpha}{\Gamma \vdash M\ N : (pqr){\cdot}\beta}\ (S_V\text{-}@)$$

(b) The Call-by-Value Probabilistic Simple Type System $\mathbf{PST}_V$

Figure 9: The Probabilistic Simple Type Systems

Theorem 6.7 *For any derivation $\Gamma \vdash M : p{\cdot}\alpha$ and any reduction $M \to^* \mathcal{M}$, there is a family of provable judgements $(\Gamma \vdash N : q_N{\cdot}\alpha)_{N \in SUPP(\mathcal{M})}$ such that*

$$p \leq \sum_N q_N{\cdot}\mathcal{M}(N)\ .$$

Theorem 6.8 *The type inference of $\boldsymbol{PST}_N$ is decidable.*

Proof. First, we infer a derivation without probabilistic annotations. Then, we use the inference algorithm for simply typed λ-calculus except that the algorithm use a non-deterministic oracle[4] that can stop the inference of a branch by the rule ($\mathbf{MI}_N$-$\langle\rangle$). Then we add the probabilistic annotations in a top-down manner. □

However, this system is far from being complete. For example, the following term is not typable (with a probability above 0): $(\lambda f.f\ (f\ \boldsymbol{I}))\ (\boldsymbol{I} \oplus \boldsymbol{\Omega})$.

As usual, the same ideas can be turned into a type system for CBV evaluation, called $\mathbf{PST}_V$ and described in Figure 9(b).

7 Related Work

Intersection types appeared for the first time in the classic work by Coppo and Dezani [7]. Although the reason for their inception was mainly semantical, they have been immediately recognised as a means to characterise various notions of normalisation in the λ-calculus. As such, they have later been the object of many studies. In particular, if type intersection is *not* assumed to be idempotent, intersection types are able to capture also quantitative properties, like the number of reduction steps to normal form [5]. Recently, intersection types have been proved to be useful in synthesis [13], but also in verification [36, 26]. Noticeably, none of the work above deal with probabilistic effects.

The denotational semantics of probabilistic higher-order programming languages have been studied since the eighties, with the pioneering work of Sahjeb-Djaromi [34], followed by many others, and in particular by Jones and Plotkin [23]. Particularly relevant to our work are probabilistic

[4]Notice that the size of the derivation is bounded by the size of the term so that there is a finite number of choices made by the oracle.

coherent spaces [10, 14]. In [14], the authors derive intersection types from probabilistic coherent spaces. Those are fundamentally different from ours, as discussed in Section 4.

Recently, probabilistic higher-order computation has received a lot of attention from the research community, given the appearance of programming languages like `Church` or `Anglican`. Those programming languages provides not only primitives for sampling from continuous distributions, but also operators for conditioning. As such, they cannot be subjected to the analysis we do here.

Designing powerful type systems for probabilistic programming languages, and for higher-order ones in particular, has remained an elusive research direction until very recently. The only publication dealing with it is due to Dal Lago and Grellois [28], and proposes a system of sized types ensuring almost-sure termination. There is however a difference: while sized types are by definition incomplete, intersection types are designed in such a way as to reflect the underlying dynamic process very precisely.

8 Conclusion

In this paper, we have showed that probabilistic higher-order languages can indeed be subjected to intersection type analysis, obtaining results similar to those one gets in the usual, deterministic, setting. The price to pay for capturing a class of terms which stands very high in the arithmetical hierarchy is the fact that not *one*, but *countably many* derivations need to be analysed for completeness to hold. The main result then trades completeness and elegance for tractability, providing a type system which is complete only up-to approximations.

Topics for further work certainly include the study of more tractable version of our type system. Applications of (variations of) our type disciplines to verification and synthesis, possibly following the works by Rehof, Kobayashi and others [13, 36, 26] are also very intriguing research directions, which however lie outside the scope of this paper.

But, for now, we intend to explore the underlying denotational semantics. Indeed, it is folklore that any reasonable intersection type system should somehow give rise to a denotational model. In the case of the monadic system, the unusual asymmetry between CBN and CBV indicates that the underlying semantics is highly non-standard and thus an interesting subject of study.

References

[1] Samson Abramsky. Abstract interpretation, logical relations and kan extensions. *Journal of Logic and Computation*, 1(1):5–40, 1990.

[2] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[3] Henk P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North Holland, 1984.

[4] Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Coupling proofs are probabilistic product programs. In *Proc. of POPL 2017*, pages 161–174. ACM, 2017.

[5] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

[6] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19(1):139–156, 1978.

[7] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

[8] Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.

[9] V. Danos and R. Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.

[10] Vincent Danos and Thomas Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 209(6):966–991, 2011.

[11] Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

[12] Karel De Leeuw, Edward F Moore, Claude E Shannon, and Norman Shapiro. Computability by probabilistic machines. *Automata studies*, 34:183–198, 1956.

[13] Andrej Dudenhefner and Jakob Rehof. Intersection type calculi of bounded dimension. In *Proc. of POPL 2017 2017*, pages 653–665, 2017.

[14] Thomas Ehrhard, Michele Pagani, and Christine Tasson. Probabilistic Coherence Spaces are Fully Abstract for Probabilistic PCF. In *Proc. of POPL 2014*. ACM, 2014.

[15] John Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[16] Jean-Yves Girard. Linear logic. *Theorical Computer Science*, 50:1–102, 1987.

[17] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[18] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *UAI*, pages 220–229, 2008.

[19] Jean Goubault-Larrecq. Full abstraction for non-deterministic and probabilistic extensions of PCF I: the angelic cases. *Journal of Logic and Algebraic Methods in Programming*, 84:155–184, 2015.

[20] Jean Goubault-Larrecq and Daniele Varacca. Continuous random variables. In *Proc. of LICS 2011*, pages 97–106, 2011.

[21] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *Proc. of LICS 2017*, pages 1–12, 2017.

[22] John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *Proc. of POPL 1996*, pages 410–423, 1996.

[23] C. Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *Proc. of LICS 1989*, pages 186–195, 1989.

[24] Neil D. Jones and Nina Bohr. Call-by-value termination in the untyped lambda-calculus. *Logical Methods in Computer Science*, 4(1), 2008.

[25] Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost-sure termination. In *Proc. of MFCS 2015*, volume 9234 of *LNCS*, pages 307–318, 2015.

[26] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proc. of LICS 2009*, pages 179–188, 2009.

[27] Jean-Louis Krivine. *Lambda-calculus, types and models.* Ellis Horwood series in computers and their applications. Masson, 1993.

[28] Ugo Dal Lago and Charles Grellois. Probabilistic termination by monadic affine sized typing. In *Proc. of ESOP 2017*, volume 10201 of *LNCS*, pages 393–419, 2017.

[29] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. of POPL 2014*, pages 297–308, 2014.

[30] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theorical Computer Science*, 228(1-2):175–210, 1999.

[31] Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.

[32] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[33] Michael O Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

[34] N. Saheb-Djahromi. Probabilistic LCF. In *Proc. of MFCS 1978*, volume 64 of *LNCS*, pages 442–451, 1978.

[35] Eugene S Santos. Probabilistic Turing machines and computability. *Proceedings of the American Mathematical Society*, 22(3):704–710, 1969.

[36] Takeshi Tsukada and Naoki Kobayashi. An intersection type system for deterministic pushdown automata. In *Proc. of TCS 2012*, pages 357–371, 2012.

[37] Daniele Varacca and Glynn Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006.